A Reliable and Realistic Approach of Advance Network Reservations with Guaranteed Completion Time for Bulk Data Transfers in Grids*

Kashif Munir
University of Innsbruck, Austria
Technikerstrasse 21a
Institute of Computer Science
Kashif.Munir@uibk.ac.at

Somera Javed
National University of Computer and Emerging Sciences, Islamabad
A. K. Brohi Road, H-11/4, Islamabad
Somera11@gmail.com

Michael Welzl
University of Innsbruck, Austria
Technikerstrasse 21a
Institute of Computer Science
Michael.Welzl@uibk.ac.at

ABSTRACT
Advance Reservation mechanisms in Grid systems should include the network just like any other resource; this is usually not only a technical, but also an administrative challenge. In this paper, we present a QoS mechanism for bulk data transfers which minimizes the necessary support from network service providers. By shifting the complexity of controlling the traffic to end nodes, where we combine admission control with congestion control, we can provide per-flow guarantees while efficiently using the available network capacity.

Keywords
QoS, Grid, Bulk Data Transfer, Advance Reservation, Resource Broker, UDT

1. INTRODUCTION
It is desirable for a Grid scheduler to have Bulk Data Transfers completed within a predefined time. To this end, the concept of "Advance Reservation" of Grid resources should include the network. This is often impossible due to administrative hurdles, as such Quality of Service (QoS) guarantees normally require all the routers along an end-to-end path to be involved, which means that they must be configured to support the requested service. From the point of view of an Internet Service Provider (ISP), this is a significant financial investment because of the manpower that is needed for installing and maintaining a complicated router configuration. Moreover, there is a certain risk associated with this setup – misconfiguration can lead to degraded service of other (non-Grid) customers, which can cause even greater financial loss.

In this paper, we describe a mechanism which strikes a balance between these two extremes for ISPs: we only require a single protected traffic aggregate from them – this is easy to install, and a variety of different network mechanisms such as DiffServ or MPLS can be used as tools for providing it. No complicated configuration is needed in the network, and per-flow guarantees are still possible, as we precisely control what enters the aggregate; this control is executed in the end systems of the Grid.

Our mechanism makes use of the distributed nature of a Grid, where we can assume that end nodes that have access to our system are trustworthy, and cooperation is in their own interest. We specify that, before using bandwidth in our protected traffic aggregate, it must be requested from a Resource Broker (RB) – a common service in Grids where one can, for instance, ask for a machine with a certain CPU power; it is our intention to extend this element with the ability to grant Advance (Network) Reservation. Moreover, we attain a deterministic behavior by prescribing the use of a single particular congestion control mechanism for all end systems.

Note that these assumptions place some fundamental limits to our mechanism: in traditional network "Bandwidth Broker" based architectures, for example, the Bandwidth Broker informs edge routers about any granted service, so as to enable them to ensure that the behavior of end nodes conforms to their promises. Our assumption of trusted end systems allows us to relax this constraint, thereby reducing the administrative burden on ISPs.

There is also the issue of resilience, which is of utmost importance in a traditional QoS scenario because it usually involves a customer who pays for the granted service. In Grids, where guarantees are sought for file transfers rather than live streams, a failure to provide an offered service will simply make an application a little slower than expected, which is a much less severe effect than, for instance, dropped frames of a video where a user paid for perfect quality. For this reason, we can base our design on the assumption that path changes are rare enough to allow our mechanism to work within reasonable bounds, and there is no need to foresee a mechanism for pinpointing a path, which would again involve routers and hence cause administrative effort on the ISP side.

* The work described in this paper is partially supported by the Higher Education Commission (HEC) of Pakistan under the doctoral fellowship program for Austria, the European Union through the FP6-IST-045256 project EC-GIN and D. Swarovski & Co.

GridNets'07, October 17–19, 2007, Lyon, France.
 ISBN 978-963-9799-07-3
DOI 10.4108/gridnets.2007.2163

Some information about the network is however needed, and would have to be communicated to our RB from a constantly active distributed measurement system in the Grid:

- Bottleneck link capacities[1] must be known for all bottlenecks of all end-to-end paths.
- Shared bottlenecks[2] must be detected.

We assume that knowledge about bottleneck capacities and shared bottlenecks is available at the end systems, and point out that there are enough indications in the literature that obtaining such measurements would be feasible. This literature will be surveyed in the next section. We explain how our mechanism works in Section 3, and support our explanations with simulation results in Section 4. Section 5 concludes.

2. RELATED WORK

2.1 Network Reservations

In general, there are two types of network resource reservations in computer networks [1]. One is immediate reservation which is made in a just-in-time manner and the other is advance reservation, which allows reserving network resources a long time before they are actually used.

Early work on advance reservation focused on reservation protocols like RSVP [2] and ST-II [3], admission control mechanism [4] and routing algorithms for networks with advance reservations [5,6].

Grid applications need guarantees of Quality of Service (QoS) [7,8]. Targeting deadline support for bulk data transfers, the problem of network resource reservation [9] has been proposed to be studied within the grid scope. An example for a Grid toolkit that supports such mechanisms is Globus with its GARA resource allocation component [10].

The issue of bandwidth fragmentation is discussed by Burchard et al. [1]. Bandwidth fragmentation may reduce acceptance percentage of requests arriving later. They propose the idea of malleable reservation to address the problem for which a start time and single rate value can be selected from a range of values.

In [12], if the latest call request is a malleable request, the method of [1] or [11] is used to adjust the bandwidth or duration to satisfy the requester. However for a fixed request, the bandwidth or duration of transmission cannot be modified and the only way to avoid being rejected is to adjust the bandwidth of admitted malleable requests. The trouble with this mechanism is the extra overhead in finding and adjusting the admitted reservations which may be modified. The Multi-Interval mechanism, presented in [13], avoids this trouble. The mechanism is based on the concept that a request should not be rejected if there is at least one feasible solution to accept it and if there are multiple solutions, the one which yields the minimum flow time is chosen and is not changed after that. The comparison of the mechanisms in [13] shows that the Multi-Interval mechanism provides the best Acceptance Percentage of requests than that of the other mechanisms.

In [14] a general view of the network resources sharing in Grids and Grids traffic isolation is presented. Optimization of bandwidth sharing among Grid flows is given [15] by manipulating the transmission windows of the flexible requests between minimum and maximum rates to maximize the acceptance rate of requests and to maximize the network utilization while still meeting their deadlines. The formulated optimization problem is proven to be NP-complete.

Two types of strategies for scheduling bulk data transfers are possible [16]. One strategy is to immediately grant or reject admission to a reservation request on its arrival time. This strategy can be called as on-demand admission control. In the other strategy, if a reservation request can not be granted or rejected at the time of its arrival, it is put in a queue to explore its possible admission later. This strategy can be called as queue-based admission control. Our mechanism, which will be explained in section 3.2.2, is based on the former, on-demand admission control, strategy.

A time-slot based approach for scheduling the elastic and streaming requests is described in [17]. However, the effect of the extra signaling overhead, which is due to the manipulation of the data transfer rates of individual flows, is not taken into account in this approach. Our mechanism considers this overhead.

A mechanism of applying fully distributed congestion control based admission control was introduced together with earlier scheduling approaches in [18]; for ease of understanding, we will briefly recapitulate this prior work in the section 3.2.1 (please see [18] for in-depth explanation of the mechanism).

Except [18], in all the above approaches, the flows send at a fixed rate during a time slot or block assuming loss-free networks and no scheduling overhead due to admission control computation. However we propose a reliable and realistic mechanism of Bulk Data Transfers in which the residual network capacity is quickly and fairly shared by all existing flows, which minimizes flows completion times and which consequently results in higher acceptance of reservation requests in the network. Our mechanism includes all extra overheads caused due to communication between senders and the RB, network control information between senders and receivers and re-transmission of lost packets. Our mechanism is not dependent on time slots. A flow can terminate at any time upon completion of its data transfer and the other admitted requests automatically share the residual capacity. We consider dedicated networks and use precise bandwidth reservation to provide QoS guarantees.

2.2 Network Measurements

The information about the network that is needed for our architecture can be obtained via an end-to-end measurement system such as the one described in [19]. This system could send probe traffic, or require the sender to cooperate by time stamping the packets or sending them back-to-back. Active methods for deducing bottleneck capacities via so-called “packet

[1] In what follows, the term “capacity” does not refer to the physical capacity of a link but the maximum transmission rate that it provides to users of the protected high-class traffic aggregate.

[2] In the context of this paper, a bottleneck is the link with the smallest capacity along a path.

pairs" have been studied for a long time, starting with [20], and led to a large number of measurement tools. An example of such tool is "NetTimer" [21]. Recently, strictly passive methods were investigated, where the fact that TCP itself sends packet pairs if receivers use "Delayed ACKs" (as the specification suggests) is exploited [22].

Detecting shared bottlenecks in the network is also not a new problem; various techniques were proposed in [23,24]. In [25], a completely passive approach for learning about shared bottlenecks was introduced.

2.3 Congestion Control

Common admission control mechanisms assume all flows to use a certain fixed (or maximum) rate. It is a key feature of our mechanism that it manages to efficiently utilize network resources in a scalable manner because flows automatically increase their rates as bandwidth becomes available. This is attained by using a congestion control mechanism for all end-to-end flows; moreover, we use a mechanism that is designed for high-speed networks (networks with a large bandwidth-delay product), where standard TCP congestion control is known not to yield satisfactory performance.

Most end-to-end congestion control mechanisms in the literature converge to a rate which depends on the round-trip time (RTT). One particular fairness measure that would suit our needs is called "max-min fairness". The authors of [26,27,28] showed that the well-known TCP variants FAST TCP, Scalable TCP (STCP), HighSpeed-TCP, BIC, CUBIC, H-TCP are not "RTT-fair". There are however exceptions: UDT [29] is designed to be max-min fair. Because it is designed for high-speeds and particularly convenient in a Grid setting, we chose UDT for our mechanism, but stress that any max-min fair congestion control mechanism could be used in its place.

3. THE QOS MECHANISM

3.1 Introduction

Our QoS mechanism provides network guarantee to a flow by admitting it with an average required rate (ARR) of x bits per second to make it possible for it to meet its deadline. After admission, a fair allocation is provided to flows using a max-min fair congestion control mechanism in such a way that at any time the rate of any flow does not go below its average rate requirement. In a Grid we can also exploit the knowledge that deadlines are sometimes known in advance, and it is important to have a network RB which can reserve flows in advance.

The QoS mechanism achieves high and fair utilization of bottleneck link's bandwidth, and it increases the acceptance of new flows in the network by minimizing the mean flow time using a high speed congestion control mechanism, UDT, which is reliable, operates purely in an end-to-end fashion and is fair. The admission and termination of a flow is controlled through the RB residing on a node in the network and by having a sender – RB signaling mechanism. Note that we only assume a single node for the sake of simplicity, and distributing the resource broker with a mechanism as in [30] would not change anything about our architecture.

3.2 Design/Operation

It is assumed that an efficient technique for measuring the bottleneck capacity and the shared bottlenecks is used in the Grid network; there are techniques which achieve that (see section 2). Further, we assume that all the QoS traffic is isolated from any other traffic – that is, the RB has complete knowledge of all flows that enter and leave the system in our QoS mechanism. All the flows in our mechanism must use the same max-min fair congestion control mechanism.

The basic idea is to divide the bandwidth into weights of some predefined rate value (e.g. y bits per second for each weight). So a flow requiring an average rate of *x* bits per second takes a certain weight of *x/y* of the bottleneck capacity. Each flow informs the RB about its desired admission in the network and it also informs the RB as soon as it terminates so that its entry is deleted by the RB and the resources owned by the flow are relinquished. The RB checks if the ARR is available to admit a new flow in the network. In the case of congestion or loss in the network the rates of all flows are reduced to their ARRs.

The following example explains the scenario. Let us assume we have a bottleneck bandwidth of 40 Gbps and we have four flows at the start with different deadlines sharing the bottleneck. Suppose each flow requires an ARR of 10 Gbps. Suppose at a time *t1* the flow-1 terminates and after sometime, at time *t2*, a new flow (flow-5) requiring an ARR of 10 Gbps wants to be admitted in the network before any of the existing flows (i.e. flows 2, 3 and 4) terminates. At the time *t1* the flows tend to quickly utilize the available capacity using the congestion control protocol.

3.2.1 ARR_CC

The mechanism is described in detail in [18]. The RB knows that there is 10 Gbps capacity available for allocation. The requirement for the new flow can be met; the new flow is then admitted in the network. Now the rate of each flow (flow 2, 3, 4 or 5) is 10 Gbps.

The pictorial representation of the 3 previous flows (flows 2, 3 and 4) and the new flow (flow 5) at the time of admission of flow-5 in the network is shown in figure 1. In the figure 1 *E2*, *E3*, *E4* and *E5* are the deadlines of the flows 2, 3, 4 and 5 respectively and *E2'*, *E3'* and *E4'* show the expected termination times due to increase of rates of flows 2, 3 and 4 respectively at time *t1*. Note that the figure 1 shows the ideal adjustment of rates and is just used to explain the example; in an actual simulation the adjustment of rates takes some time according to the transport protocol.

At the time *t2*, flow-5 could not have been admitted in the network if its ARR was more than 10 Gbps because at that particular moment the RB after summing the ARRs of all the existing flows could only guarantee the availability of 10 Gbps for a new flow.

Note that, after *t2*, the existing flows could theoretically transfer at a smaller rate than their ARR as they were transferring data at a higher rate prior to the admission of flow-5. The new mechanism, described in section 3.2.2, tries to take advantage of this fact by reducing the ARRs of flows below their initially calculated ARRs for dynamic admission control.

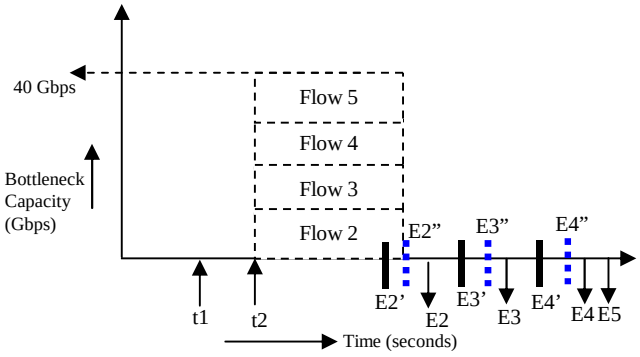

Figure 1. Flow-5 is admitted. Flows 2, 3 and 4 decrease their rates to their ARR i.e. 10 Gbps. The dotted marks along the x-axis show the expected completion times *E2", E3"* and *E4"* for the termination of flows 2, 3 and 4 respectively according to the rate of the flows at *t2*.

The Admission Control Algorithm of *ARR_CC* as mentioned in [18] is given below.

D_S, D_F, T_S, ARR, ID, R_T: Data size, duration, start time, ARR, ID and reservation type of a reservation request
$R_T \in$ {IR,AR}: IR = Immediate reservation, AR = Advance reservation
Record of a flow: {T_S, T_E, D_F, D_S, ARR, ID}
Φ: Set of records of the currently accepted flows sharing the bottleneck link
C_T: The total capacity of the bottleneck link
T_C: The current time

```
Procedure Admission_Control_ARR_CC(Network_Info)

While (All flows are processed)

If (a new reservation is requested)

      ARR = ⌈D_S/D_F⌉
      ID = generate ID for new request

      If (Admission(Φ,ID,ARR,R_T,D_F,D_S,T_S,T_C,C_T)
                = YES) Then
           {Start the flow with its ARR at its start
           time using a max-min fair congestion control
           protocol}

      Else
           {Reject the flow}


If (a served request is completed) Then
        Termination(Φ,ID)

End While

End Procedure

Procedure Admission(Φ,ID,ARR,R_T,D_F,D_S,T_S,T_C,C_T)

Set C_R to 0
// C_R is the reserved capacity

If (R_T = IR) Then
// Immediate reservation request
        T_S = T_C
        T_E = T_C + D_F
        // T_E is the end time of a flow

Else
// Advance reservation request
        T_E = T_S + D_F
For Each flow ∈ Φ
  If ((Φ(flow).T_S < T_E) AND (Φ(flow).T_E > T_S)) Then
      C_R = C_R + Φ(flow).ARR
End For

If (C_T – C_R) > ARR Then
        Φ = Φ + flow
//flow = Flow_Record(T_S,T_E,D_F,D_S,ARR,ID)
        Return "Yes"
Else
        Return "No"

End Procedure


Procedure Termination (Φ,ID)

For Each flow ∈ Φ
        If (Φ(flow).ID = ID) Then
                Φ = Φ - flow
                Break

End For

End Procedure
```

3.2.2 ARR_Adjustable_CC

In this mechanism the RB contacts each flow for its updated ARR only when a new flow which is seeking admission in the network does not find its ARR available. Each flow then updates its ARR and sends a message to the RB to update the RB about its ARR depending on the data left to be transferred and the remaining time left to meet the flow's deadline. The Admission Control Algorithm of this mechanism for the RB is given below.

The inputs are same as that in ARR_CC

```
Procedure Admission_Control_ARR_Adjustable_CC
(Network_Info)

While (All flows are processed)

   If (a new reservation is requested)

      ARR = ⌈D_S/D_F⌉
      ID = {generate ID for new request}

       If (Admission(Φ,ID,ARR,R_T,D_F,D_S,T_S,T_C,C_T)
       = YES) Then
         {Start the flow with its ARR at its start time
         using a max-min fair congestion control
         protocol}


       Else
          For Each flow ∈ Φ
            If (T_C > Φ(flow).T_S) Then
            // The ARR of the AR which is not started
            // yet must not be updated
                  Φ(flow).D_S = {Remaining data size of
                  the flow to be transferred}
                  Φ(flow).D_F = Φ(flow).T_E – T_C
                  Φ(flow).ARR = ⌈Φ(flow).D_S/Φ(flow).D_F⌉
          End For

          If (Admission(Φ,ID,ARR,R_T,D_F,D_S,T_S,T_C, C_T) =
          YES) Then
                {Start the flow with its ARR at its
                start time using a max-min fair
                congestion control protocol}
          Else
                {Reject the flow}
```

```
If (a served request is completed) Then
        Termination(Φ,ID)

End While

End Procedure
```

The Admission and Termination procedures are same as in ARR_CC.

3.3 Implementation

One of the key components of our proposed QoS mechanism is the RB which is designed for admission control and to maintain the current state of network (i.e. all information about existing flows, shared bottleneck links and their capacities, paths etc). After the admission of a flow with its ARR, the transfer rate starts to increase according to UDT's congestion control mechanism. Upon completing the transfer of a flow, the sender sends a termination message to the RB. This message passing takes only a few milliseconds on average, which is quite negligible as compared to a typical Grid flow transfer time in which a huge amount of data is transferred. In the simulations the FTP application protocol is used over the UDT high-speed data transfer protocol.

4. PERFORMANCE EVALUATION

A single bottleneck link dumbbell network configuration is used for the simulations using ns-2. The bottleneck capacity is 1 Gbps and the bottleneck delay is set to 50ms. Drop Tail routers are used. The buffer size of the bottleneck link is set to 100% of Bandwidth-Delay product. The packet size is set to 1500 bytes. The capacity of side links is 10 Gbps and the delay of each side link is set to 2ms.

In the experiments, 5 sets of 10 simulations are performed. In each simulation 100 flows are run. Within each set the size of all flows is the same, however the size of a flow varies from 500 MB to 1500 MB from one set to the other. The average inter arrival time of flows is 4 seconds and the transfer duration of each flow is 50 seconds. For each simulation within a set, there are mixed types of randomly generated reservation requests, immediate and advance reservations. In each simulation the arrival time of a new reservation is also randomly chosen in each 4 seconds interval. The start time of an advance reservation request is also selected randomly in the interval [25,75] relative to the arrival time of the flow. The data size and the duration of its transfer are same for all flows. The results of the experiments are shown in figures 2, 3 and 4. Each block in figures 2 and 3 and each point in figure 4 represents an average of the results of 10 simulations of a set. The standard deviation of the results of all sets is less than 2, which is quite small as compared to the possible range of results.

We have compared the results of our mechanism with *ARR_CC*, with a *Fixed-Rate* transfer mechanism and with *Multi-Interval* mechanism. A *Fixed-Rate* transfer mechanism represents a traditional QoS architecture such as IntServ/RSVP. In a *Fixed-Rate* mechanism, a flow can only send its data at a constant ARR and completes on its deadline. The *Multi-Interval* mechanism is based on the *Greedy-Accept* and the *Minimize-FlowTime* heuristics presented in [13]. As mentioned in [13], "Greedy-Accept means: If there is at least one feasible solution to accept a coming request, the request should not be rejected. And Minimize-FlowTime means: If there are multiple feasible solutions in the solution space, the one with minimal completion time will be chosen". The scheduling of rates for a flow in different time slots in the *Multi-Interval* mechanism is determined at the moment the request arrives, and is not changed after that.

In *Fixed-Rate*, *ARR_CC* and *ARR_Adjustable_CC* the immediate reservations start at the time of the arrival of the request whereas in *Multi-Interval* the immediate reservations can start at anytime after their arrival as long as they meet their deadlines.

Figure 2 shows that the acceptance percentage of *Fixed-Rate* is less than that of *ARR_Adjustable_CC* and *Multi-Interval*. The acceptance percentage of *Fixed-Rate* is less than that of ARR_CC when the network is lightly loaded however the acceptance percentage of *Fixed-Rate* is more than *ARR_CC* when the network is heavily loaded. The mean flow time of *Fixed-Rate* is significantly higher than that of the other mechanisms as shown in figure 3. This is due to the reason that the flows in *Fixed-Rate* can only transfer their data at their ARRs and do not take increase their rates to take advantage of the available capacity during any time slot.

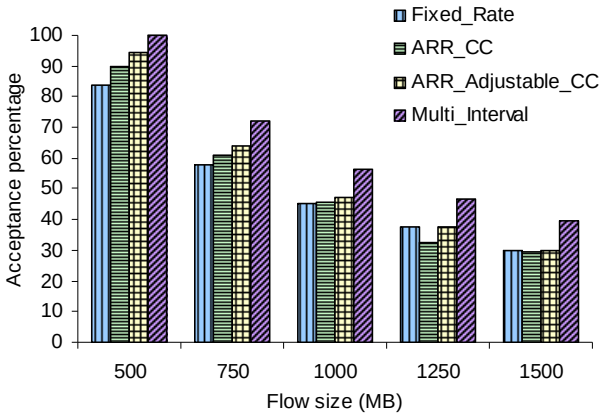

Figure 2. Acceptance percentage of flows. The load on the network is increased by increasing the data size of each flow.

As expected *Multi-Interval* has the best acceptance percentage and has the least mean flow time than that of the all other mechanisms. This is due to the assumption of ideal networking conditions and no communication and computation delays and also because it is not a reliable and realistic mechanism of data transfer. In *Multi-Interval* if there is at least one flow in the network at any moment, the residual network capacity is not wasted.

In the other two mechanisms (*ARR_CC* and *ARR_Adjustable_CC* which are reliable and practical mechanisms), *ARR_Adjustable_CC* has better Acceptance percentage than that of *ARR_CC*. This is due to the reason that *ARR_Adjustable_CC* tends to accommodate a new flow by decreasing ARRs of existing flows. However *ARR_CC* has smaller mean flow time than that of *ARR_Adjustable_CC*. Due to the accommodation of a new reservation request in

A*RR_Adjustable_CC* the existing flows decrease their ARRs which consequently increase the flow time.

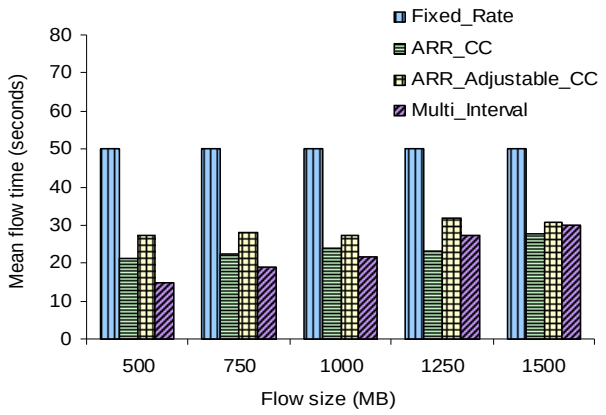

Figure 3. Mean flow time of flows. The load on the network is increased by increasing the data size of each flow.

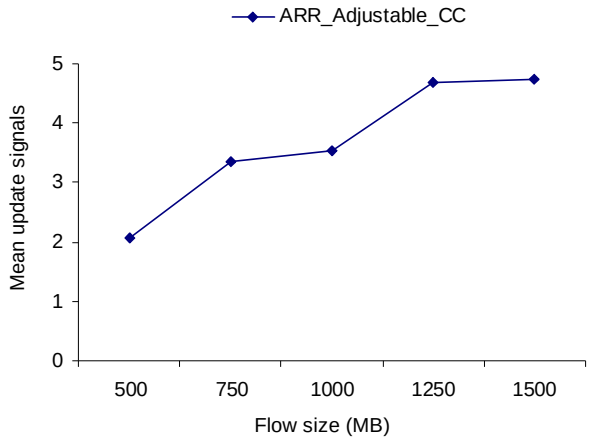

Figure 4. Mean number of update signals per flow in *ARR_Adjustable_CC*. The load on the network is increased by increasing the data size of each flow.

The ability of *ARR_Adjustable_CC* to accommodate some of the new reservation requests even if the bandwidth demand can not be fulfilled comes at a cost of the signaling overhead due to the update signals which are sent by the existing flows to the RB. The existing flows adjust their ARRs according to their remaining data sizes and the durations which are left for their transfer. Figure 4 shows the mean update signals of *ARR_Adjustable_CC*. The curve shows that the mean number of update signals per flow increases as the load on the network increases. This is due to the reason that with increase in the load on the network the acceptance percentage decreases and consequently the signaling to the senders to adjust their ARRs for the possible adjustment of a new reservation request also increases.

5. CONCLUSION AND FUTURE WORK

Our results show that, by using a fair and a stable congestion control mechanism like UDT and by using admission control to provide network reservation guarantees for elastic flows, the network can be fully utilized, resulting in early completion of a long-lived flow which consequently enables us to admit more flows earlier than it would have been possible without using our mechanism. Our contribution is that we have shown the design and the implementation of *ARR_Adjustable_CC*, which is reliable and realistic and it considers the computation and communication overheads and delays. ARR_Adjustable_CC is an on-demand admission control strategy as it accepts or rejects a reservation request on its arrival time.

Moreover, in our mechanism, some of the new requests can be accepted even if the required bandwidth is not available. This is done at the cost of decreasing the rates of some already existing flows even below their ARRs, and results in an even higher admission percentage of new flows. As we have seen, this benefit has resulted in slightly increased mean flow times than the other reliable and on-demand admission control strategy, *ARR_CC*. However, since the main goal of our system is to admit as many flows as possible while keeping all their deadlines, we consider this disadvantage to be of minor relevance.

In future, we will evaluate our mechanism on a bigger network topology of multiple bottlenecks.

6. REFERENCES

[1] Burchard, L., Heiss, H., Rose, D. *Performance issues of bandwidth reservations for grid computing*. Proceedings of Computer Architecture and High Performance Computing, (November 2003), pp 82–90.

[2] Schill, A., Breiter, F., Kuhn, S. *Design and Evaluation of an Advance Reservation Protocol on Top of RSVP*. In IEIP 4th International Conference on Broadband Communications (BC '98), Stuttgart, Germany, IFIP Conference Proceedings 121, Chapman & Hall, (1998) 23–40.

[3] Reinhardt, W. *Advance Resource Reservation and its impact on Reservation Protocols*. In Proceedings of Broadband Islands '95, Dublin, Ireland, 1995.

[4] Ferrari, D., Gupta, A., Ventre, G. *Distributed Advance Reservation of Real-Time Connections*. Multimedia Systems, Vol.5. Springer-Verlag, Berlin Heidelberg New York (1997) 187-198.

[5] Guerin, R., Orda, A. *Networks with Advance Reservations: The Routing Perspective.* In: Proc. of IEEE INFOCOM 2000. 1 (2000) 118–127.

[6] Burchard, L. *Source Routing Algorithms for Networks with Advance Reservation*. Technical Report No. 2003-2, TU Berlin. Available online at http://kbs.cs.tu-berlin.de /publications/res_mgnt/tr-2003-03.pdf.

[7] Zhang, H., Keahey, K., Allcock, B. *Providing Data Transfer with QoS as Agreement-Based Service*. International Conference on Services Computing (SCC 2004), Shanghai, China, September 15 - 18, 2004.

[8] Foster, I., Roy, A., Sander, V. *A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation*. 8th International Workshop on Quality of Service, June 2000, (IWQoS 2000), pp. 181–188.

[9] Foster, I., Fidler, M., Roy, A., Sander, V., Winkler, L. *End-to-end quality of service for high-end applications*. Computer Communications, vol. 27, no. 14, pp. 1375–1388, 2004.

[10] Foster, I., Kesselman, C., Lee, C., Lindell, R., Nahrstedt, K., Roy, A. *A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation*. In 7th International Workshop on Quality of Service (IWQoS), London, UK, pages 27–36, 1999.

[11] Xing, J., Wu, C., Tao, M., Wu, L., Zhang, H. *Flexible Advance Reservation for Grid Computing*. GCC 2004: 241-248.

[12] Wu, L., Xing, J., Wu, C., Cui, J. *An Adaptive Advance Reservation Mechanism for Grid Computing*. PDCAT 2005: 400-403, 2005.

[13] Chen, B. B., Primet, P. *Supporting bulk data transfers of high-end applications with guaranteed completion time*. IEEE ICC2007 International conference on computer communication.

[14] Primet, P., Zeng, J. *Traffic Isolation and Network Resource Sharing for Performance Control in Grids*. ACNS'05, USA, October 2005.

[15] Marchal, L., Primet, P., Robert, Y., Zeng, J. *Optimal Bandwidth Sharing in Grid environment*. IEEE HPDC, Paris, France, June 2006.

[16] Kaushik, N., Figueira, S., Chiappari, S. *Flexible Time-Windows for Advance Reservation in LambdaGrids*. ACM SIGMETRICS/Performance, Saint-Malo, France, 2006.

[17] Naiksatam, S., Figueira, S. *Elastic Reservations for Efficient Bandwidth Utilization in LambdaGrids*. Elsevier's FGCS - The International Journal of Grid Computing: Theory, Methods and Applications, vol. 23, issue 1, pp. 1-22, 2007.

[18] Munir, K., Javed, S., Welzl, M., Ehsan, H., Javed, T. *An End-to-End QoS Mechanism for Grid Bulk Data Transfer for Supporting Virtualization* IEEE/IFIP International Workshop on End-to-end Virtualization and Grid Management (EVGM 2007), held as part of Manweek 2007, San Jose, California, USA, October 2007.

[19] Yousaf, M. M., Welzl, M. *A Reliable Network Measurement and Prediction Architecture for Grid Scheduling*. IEEE/IFIP International Workshop on Autonomic Grid Networking and Management (AGNM'05), Barcelona, Spain, 28 October 2005.

[20] Keshav, S. *Congestion Control in Computer Networks*. (http://blizzard.cs.uwaterloo.ca/keshav/home/Papers/data/91/thesis/keshav.th.tar.Z) PhD Thesis, published as UC Berkeley TR-654, September 1991.

[21] Lai, K., Baker, M. "Nettimer: *A Tool for Measuring Bottleneck Link Bandwidth*. In Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems, San Francisco, California, March 2001.

[22] Barz, C., Frank, M., Martini, P., Pilz, M. *Receiver-Based Path Capacity Estimation for TCP*. In Proceedings of KIVS'05, Kaiserslautern, Germany, February/March 2005.

[23] Shriram, A., Kaur, J. *Identifying Bottleneck Links Using Distributed End-to-end Available Bandwidth Measurements*. In the First ISMA Bandwidth Estimation Workshop (BEst'03), San Diego, CA, December 2003.

[24] Kim, M. S., Kim, T., Shin, Y., Lam, S., Powers, E. J. *A Wavelet-Based Approach to Detect Shared Congestion*. In Proceedings of ACM SIGCOMM 2004, August 2004.

[25] Katabi, D., Bazzi, I., Yang, X. *A passive approach for detecting shared bottlenecks*. In Proceedings of the 10th IEEE International Conference on Computer Communications and Networks, October 2001.

[26] Li, Y. T., Leith, D., Shorten, R. *Experimental Evaluation of TCP Protocols for High-Speed Networks*. Technical report, Hamilton Institute, 2005.

[27] Tan, K., Song, J., Zhang, Q., Sridharan, M. *Compound TCP: A Scalable and TCP-friendly Congestion Control for High-speed Networks*. In 4th International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet), 2006, Nara, Japan.

[28] Ha, S., Kim, Y., Le, L., Rhee, I., Xu, L. *A Step toward Realistic Performance Evaluation of High-Speed TCP Variants*. PFLDNet 2006, Nara, Japan.

[29] Gu, Y., Grossman, R.: *UDT UDP-based data transfer for high-speed wide area networks*. Computer Networks, special issue on Hot topics in transport protocols for very fast and very long distance networks, January 2007.

[30] Müller, J. A., Hessler, S., Irmscher, K. *Class of Service Concepts in Autonomous Systems*. Terena networking conference 2004 (Terena2004), Rhodes, Greece.